Expert Systems with Applications xx (2005) 1–13

Expert Systems with Applications

www.elsevier.com/locate/eswa

Improving rule sorting, predictive accuracy and training time in associative classification

Fadi Thabtah [a,*], Peter Cowling [b], Suhel Hammoud [c]

[a] *Department of Computing and Engineering, University of Huddersfield, UK*
[b] *Department of Computing, MOSAIC Research Centre, University of Bradford, Great Horton Road, Bradford BD7 1DP, UK*
[c] *Department of Electronic and Computer Engineering, Brunel University, London, UK*

Abstract

Traditional classification techniques such as decision trees and RIPPER use heuristic search methods to find a small subset of patterns. In recent years, a promising new approach that mainly uses association rule mining in classification called associative classification has been proposed. Most associative classification algorithms adopt the exhaustive search method presented in the famous Apriori algorithm to discover the rules and require multiple passes over the database. Furthermore, they find frequent items in one phase and generate the rules in a separate phase consuming more resources such as storage and processing time. In this paper, a new associative classification method called Multi-class Classification based on Association Rules (MCAR) is presented. MCAR takes advantage of vertical format representation and uses an efficient technique for discovering frequent items based on recursively intersecting the frequent items of size n to find potential frequent items of size $n+1$. Moreover, since rule ranking plays an important role in classification and the majority of the current associative classifiers like CBA and CMAR select rules mainly in terms of their confidence levels. MCAR aims to improve upon CBA and CMAR approaches by adding a more tie breaking constraints in order to limit random selection. Finally we show that shuffling the training data objects before mining can impact substantially the prediction power of some well known associative classification techniques. After experimentation with 20 different data sets, the results indicate that the proposed algorithm is highly competitive in term of an error rate and efficiency if compared with decision trees, rule induction methods and other popular associative classification methods. Finally, we show the effectiveness of MCAR rule sorting method on the quality of the produced classifiers for 12 highly dense benchmark problems.
© 2005 Elsevier Ltd. All rights reserved.

Keywords: Associative classification; Association rule discovery; Classification; Data mining; rule sorting

1. Introduction

Traditional classification rule learning approaches for building classifiers, may be divided into three main categories, i.e. divide-and-conquer (Quinlan, 1987), separate-and-conquer (Furnkranz, 1999) and covering algorithms (Cendrowska, 1987). The majority of traditional classification techniques use heuristic-based strategies for building the classifier (Witten & Frank, 2000). In constructing a classification system, they look for rules with high accuracy. Once a rule is created, they delete all positive training objects associated with it. Thus, these methods often produce a small subset of rules, and may miss detailed rules that might play an important role in some cases. The heuristic methods that are employed by traditional classification techniques often use domain independent biases to derive a small set of rules, and therefore, rules generated by them are different in nature and more complex than those that users might expect or be able to interpret (Pazzani, Mani, & Shankle, 1993).

Association rule discovery and classification are analogous tasks in data mining, with the exception that classification main aim is the prediction of class labels, while association rule mining discovers associations between attribute values in a data set. In the last few years, association rule mining has been successfully used to build accurate classifiers, which resulted in a new approach coming to life, known as associative classification (AC) (Ali, Manganaris, & Srikant, 1997; Liu, Hsu, & Ma, 1998). Empirical studies (Antonie, Zaïane, & Coman, 2003; Liu et al., 1998; Yin & Han, 2003) showed that AC often builds more accurate classifiers than traditional classification techniques and many of the rules found by AC methods can not be discovered by traditional classification algorithms.

Given a history or a training data set, the task of an AC algorithm is to discover the complete class association rules

* Corresponding author. Tel.: +44 1274 233948; fax: +44 1274 235856.
E-mail addresses: fabdelja@bradford.ac.uk (F. Thabtah), p.i.cowling@bradford.ac.uk (P. Cowling), suhel.hammoud@brunel.ac.uk (S. Hammoud).

0957-4174/$ - see front matter © 2005 Elsevier Ltd. All rights reserved.
doi:10.1016/j.eswa.2005.09.039

with significant supports and high confidences (attributes values that have frequencies above user specified constraints, denoted by minimum support (*minsup*) and minimum confidence (*minconf*) thresholds). One subset of the generated class association rules is chosen to build an automatic classifier that could be used to predict classes of previously unseen data. Unlike the classic classification approaches such as rule induction and decision trees, which construct usually small sized classifiers, AC explores all associations between attributes values and their classes in the training data set, aiming to construct larger sized classifiers.

However, since most AC techniques adopt the exhaustive search method presented in the Apriori algorithm and without pruning (Agrawal and Srikant, 1994) to discover the rules, they suffer from inherited problems such as efficiency (Liu, Ma, & Wong, 2001; Zaki, Parthasarathy, Ogihara, & Li, 1997). Furthermore, most of the current AC techniques rank rules mainly in terms of their confidence levels. When several rules have identical confidence or support, CBA algorithm (Liu et al., 1998) for example chooses randomly between them, which may degrade accuracy. This deterioration happens especially in cases where the chosen rule does not have a large representation in the training data.

For example, if we mine the 'heart' data set downloaded from (WEKA, 2000) using the CBA algorithm and without prunning with *minsupp* of 2% and *minconf* of 40%, there are 4791 potential rules with identical confidence, from which 4297 of them have the same support and only 884 of those have breaks. The CBA algorithm ranks the remaining 3383 potential rules randomly where the majority of these are not the optimal choices, and thus, the accuracy of the classifier may slightly deteriorate. There is a critical need for additional tie breaking conditions in the selection process of the rules in order to greatly reduce the chance of randomisation.

This paper aims to improve upon the prediction accuracy and efficiency of AC techniques by dealing directly with three problems. The first one is to find an effective rule selection method during the ranking process of the rules in the hope of reducing error by minimising rule random selection. The second problem deals with using a truly random seed when partitioning the training data in cross validation and ensuring that data are not artificially prepared. The third and most important issue is to develop an efficient method for rules discovery. As a result, a new AC algorithm, called Multi-class Classification based on Association Rule (MCAR) is proposed. The proposed algorithm deals with continuous attributes (where they are real or integer) as well as categorical attributes (meaning they take a value from a finite set of possible values), performs a simple shuffling on the training data before mining and uses different random seeds in cross validation, which gives a more truly error rate each time a data set is mined.

In addition, MCAR improves upon current AC methods that rank rules mainly in terms of their confidence levels by imposing more detailed conditions such as class distribution frequency and others adopted from different artificial intelligence techniques like rule cardinality for breaking ties. Also, to speed up the process of finding rules, MCAR employs an efficient intersection method that requires a single scan over the training data set, instead of using the Apriori multiscan approach. The classifier of our algorithm is of the form $\langle r_1, r_2,\ldots,r_n$, default class$\rangle$, where r_i is a rule and the default class is the majority class of the remaining unclassified instances in the training data set.

Basic concepts in AC and common algorithms are surveyed in Sections 2 and 3, respectively. Our proposed algorithm is presented in Section 4 where details about rule discovery, rule generation, evaluation phase, rule sorting and prediction of test data objects are discussed. Extensive experimental results and comparisons are demonstrated in Section 5 and finally presented, are conclusions in Section 6.

2. Associative classification

AC is a special case of association rule mining in which only the class attribute is considered in the rule's consequent, for example in a rule such as $X \rightarrow Y$, Y must be a class attribute. Let us define the classification problem in an association rule framework. The training data set T has m distinct attributes $A_1, A_2,\ldots,A_m$ and C is a list of class labels. Attributes could be categorical or continuous. In the case of categorical attributes, all possible values are mapped to a set of positive integers. For continuous attributes, any discretisation method can be used.

Definition 1. A row or a training object in T can be described as a combination of attribute names A_i and values a_i, plus a class denoted by c_j.

Definition 2. An item can be described as an attribute name A_i and value a_i.

Definition 3. An itemset can be described as a set of items contained in a training object.

Definition 4. A *ruleitem* r is of the form $\langle$*itemset*, $c\rangle$, where $c \varepsilon C$ is the class.

Definition 5. The actual occurrence (*actoccr*) of a *ruleitem* r in T is the number of rows in T that match the itemsets defined in r.

Definition 6. The support count (*suppcount*) of *ruleitem* r is the number of rows in T that match r's itemsets, and belong to a class c_i for r.

Definition 7. An itemset i passes the *minsupp* threshold if $(|i|/|T|) \geq minsupp$, where $|i|$ is the number of rows in T that contain i and $|T|$ is the number of instances in T.

Definition 8. A *ruleitem* r passes the *minsupp* threshold if $(suppcount(r)/|T|) \geq minsupp$, where $|T|$ is the number of instances in T.

Definition 9. A *ruleitem* r passes *minconf* threshold if $(suppcount(r)/actoccr(r)) \geq minconf$.

Definition 10. Any itemset i that passes the *minsupp* threshold is said to be a frequent itemset.

Definition 11. Any *ruleitem r* that passes the *minsupp* threshold is said to be a frequent *ruleitem*.

Definition 12. An actual class association rule is represented in the form: $(A_{i1},a_{i1})\wedge(A_{i2},a_{i2})\wedge\ldots\wedge(A_{1m},a_{im})\rightarrow c_j$, where the antecedent of the rule is an itemset and the consequent is a class.

A classifier is of the form $H{:}A_1,A_2,\ldots A_n\rightarrow Y$, where A_i is an attribute and Y is the class. The main task of AC is to construct a set of rules (model) that is able to predict the classes of previously unseen data, known as the test data set, as accurately as possible. Formally, the goal is to find a classifier $h\varepsilon H$ that maximises the probability that $h(a)=y$ for each test instance (a, y).

3. Related work

One of the first algorithms to use an association rule mining approach for classification was proposed in (Liu et al., 1998) and named CBA. CBA implements the famous Apriori algorithm (Agrawal & Srikant, 1994) in order to discover frequent *ruleitems*. Once the discovery of frequent *ruleitems* is finished, CBA proceeds by converting any frequent *ruleitems* that passes *minconf* into a rule. In so doing, only one subset of the generated classification rules will be considered in the final classifier. Evaluating all the generated rules against the training data set does the selection of the subset. The frequent *ruleitems* discovery and rules generation are implemented in two separate phases in CBA.

The Apriori algorithm has been applied to misleading classification data in the water acoustic signals field (Huange, Zhao, & Xie, 1997). The goal is to separate the misclassification region of the feature space and to extract useful rules for correct prediction in that misclassification region. A two-phase training algorithm has been presented to exploit association rules in order to reveal intrinsic rules that could lead to a correct classification. Experimental tests on 964 training examples contained in real vessel data indicated the potential use of association rule discovery approach in improving the accuracy for the difficult to classify underwater acoustic signals.

An AC algorithm that selects and analyses the correlation between high confidence rules, instead of relying on a single rule, has been developed in (Li, 2001). It uses a set of related rules to make a prediction by evaluating the correlation among them (Li, 2001). The correlation measures how effective the rules are based on their support values and class distribution. In addition, a new prefix tree data structure named a CR-tree is used to handle the set of rules generated and to speed up the rule retrieval process.

Improvements upon the CBA algorithm with regards to using multiple supports and extracting long patterns were reported in (Liu et al., 2001). These improvements resulted in a multiple local supports technique for the discovery of frequent *ruleitems* for not only the dominant classes but also for the low frequency ones. In order to extract long patterns from data, a hybrid algorithm that combines CBA with decision trees (C4.5) (Quinlan, 1993) has also been presented. The basic idea of the hybrid algorithm is to segment the training data, and to apply the CBA algorithm on each segment. Finally, the classifier that achieves the highest accuracy on one of the segments is selected to classify new unseen data objects.

The problems of minimising the number of rules in AC and the effectiveness of the rule-based classifier on missing values in test data sets have been studied in (Li, Han, & Pei, 2001). The authors introduced two simple methods, one extends C4.5, and the other adopts the CBA algorithm. The presented AC method aims to derive smaller set of rules than that of CBA. This set is claimed to be able to predict test data objects as accurately as a CBA classifier.

A method based on association rule for medical image classification has been presented in (Antonie et al., 2003). It consists of three major steps, in which the first step involves cleaning up the medical images and extracting target features. Step two uses CBA approach in order to learn rules, which are used to build the classifier in step three.

A new approach for building classification systems based on both positive and negative rules has been introduced in (Antonie & Zaïane, 2004). The ‘interestingness’ of the rules for the proposed algorithm is based on the correlation coefficient that measures the strength of the linear relationship between a pair of variables. Besides confidence and support thresholds, correlation coefficient has been used for pruning the final classifier, giving a much reduced rules set if compared with support and confidence pruning methods. The algorithm in (Antonie and Zaïane, 2004) finds the frequent *ruleitems* using Apriori approach and ranks the rules using CBA rules ranking method.

Our proposed algorithm uses the core concepts of association rule mining in classification framework. However, MCAR has many distinguishing features over other AC algorithms that will be discussed in Section 4.6.

4. MCAR algorithm

The algorithm proposed in this paper consists of two main phases: rule generation and a classifier builder. In the first phase, the training data set is scanned once to discover frequent one *ruleitems*, and then MCAR recursively combines *ruleitems* generated to produce potential frequent *ruleitems* (candidate *ruleitems*) involving more attributes. The supports and confidences for candidate *ruleitems* are calculated simultaneously, where any *ruleitem* with support and confidence larger than *minsupp* and *minconf*, respectively, is created as a potential rule. We will explain in details the discovery of frequent *ruleitems* step in Section 4.1. In the second phase, rules created are used to build a classifier by considering their effectiveness on the training data set. Only effective rules will be kept in the final classifier. Fig. 1 represents the proposed algorithm, which we will explain in details below.

Data used by MCAR contain a header that indicates file name, attribute names, and number of rows. Values for each attribute are comma-separated, and the class attribute must be the last column in the header file. Missing values in the training data set are treated as other existing attributes values.

MCAR treats not only categorical data but also continuous data. For categorical attributes, we assume that all possible values are mapped to a set of positive integers. For continuo-valued attributes, the multi-interval discretisation technique (Fayyad & Irani, 1993) has been implemented. We briefly summarise the process of discretising continuous attributes from (Fayyad & Irani, 1993). First, the training instances for each continuous attribute are sorted in an ascending form and the class values associated with each instance is given. The next step is to place break points whenever the class value changes and to calculate the information gain for each possible break point. The information gain represents the amount of information required to specify values of the classes given a breaking point. Finally, the break point that leads to the least information value is selected and the algorithm is invoked again on the lower range of that attribute until a stopping condition is met.

4.1. Frequent ruleitems discovery and rule generation

4.1.1. Training data format

Most of the works conducted in association rule discovery, e.g. (Agrawal and Srikant, 1994; Brin, Motwani, Ullman, & Tsur, 1997; Han, Pei, & Yin, 2000; Park, Chen, & Yu, 1995), have used the classic horizontal format presented in the Apriori approach. There are few association rule mining techniques which have utilised vertical format, e.g. (Savasere, Omiecinski, & Navathe, 1995; Zaki & Gouda, 2003; Zaki et al., 1997). The database in the horizontal format consists of a group of transactions, where each one has a unique identifier followed by a list of items contained in that transaction. There are several drawbacks resulting from using horizontal data representation, including multiple data scans when searching for frequent itemsets at each level and the computational cost during the support counting process of candidate itemsets.

The database in the vertical format comprises a collection of items, where each item is followed by a list of row identifiers (tids), which contain that item, this list is often called a tid-list. Empirical studies (Zaki et al., 1997; Zaki & Gouda, 2003) show that vertical layout is more efficient way of representing the data due to often, no candidate generation and support counting involved, rather items support is performed by fast intersections between tid-lists, saving huge I/O time.

For these reasons, we have used vertical layout to represent classification data sets in the proposed algorithm. To the best of our knowledge there are no AC techniques that use vertical layout for data representation or tid-lists intersections for finding frequent *ruleitems*.

```
Input: Training data (D), minsupp and minconf thresholds
Output: A set of rules

Preprocessing phase
If D contains real/integer attributes
  Discretise continuous columns using multi-interval
  discretisation method
Shuffle randomly the training objects locations

The Algorithm
Scan D for the set S of frequent one-ruleitems
Generate one-ruleitems rules
Do
   For each pair of disjoint items $I_1, I_2$ in S
   Intersect the sets of rowIds of $I_1$, and $I_2$ and store it in
   Ts
     If Ts size < itemsupp then prune the new item
    else
begin
      If (<$I_1 \cup I_2$>) passes the minsupp threshold
begin
       if (<$I_1 \cup I_2$>, $c_i$) passes the minconf threshold
   begin
          Generate a rule for (<$I_1 \cup I_2$, $c_i$>)
           $S \leftarrow S \cup <I_1 \cup I_2>$
     end if
      else discard the new ruleitem.
end if
   end else
Until no ruleitems which pass minsupp are found
end
Rank all rules generated according to confidence,
support, etc
Evaluate produced rules on D to remove useless ones
Remove all rules $I' \rightarrow c'$ from S where there is some
rule $I \rightarrow c$ of a higher rank and $I \subseteq I'$.
```

Fig. 1. MCAR algorithm.

4.1.2. MCAR intersection method

Most of the current AC techniques, including CBA, ADT (Wang, He, & Cheung, 2001) and the negative-rules (Antonie and Zaïane, 2004), adopt the Aprioi candidate generation step for the discovery of frequent *ruleitems*. Other AC algorithms such as CMAR (Li et al., 2001) and L^3 (Baralis & Torino, 2002) use the FP-growth approach (Han et al., 2000) in order to decrease the number of database passes. The main drawback in terms of mining efficiency of almost all the current AC techniques is that they make more than one pass over the training data set to discover frequent *ruleitems*, which causes high I/O overheads (Li et al., 2001; Liu et al., 2001).

In order to improve the efficiency of frequent *ruleitems* discovery, MCAR employs a technique that extends the tid-lists intersection methods of (Savasere et al., 1995; Zaki et al., 1997) to handle classification benchmark problems. Using the tid-list for each itemset in association rule mining is a good approach since the cardinality of the itemset tid-list divided by the total number of the transactions gives the support for that itemset. However, tid-lists intersection methods presented in association rules discovery need to be modified in order to treat classification problems, where classes associated with each itemset are considered when computing the support.

The frequent *ruleitems* discovery method employed by MCAR scans the training data set once to count the occurrences of itemsets of size one, from which it determines those that hold enough support. This is the first pruning performed by MCAR to discard any itemset that has a support below the user *minsupp* threshold. Another pruning occurs to discard any *ruleitem* that is not frequent (that's when the number of occurrences of an itemset and its largest frequency

Table 1
Training data 1

Rowed	A_1	A_2	Class
1	x_1	y_1	c_1
2	x_1	y_2	c_2
3	x_1	y_1	c_2
4	x_1	y_2	c_1
5	x_2	y_1	c_2
6	x_2	y_1	c_1
7	x_2	y_3	c_2
8	x_1	y_3	c_1
9	x_2	y_4	c_1
10	x_3	y_1	c_1

class is less that the *minsupp* threshold). Once frequent one-*ruleitems* are determined, we store them along with their locations (rowIds) and their classes inside arrays. Then, by intersecting the rowIds of frequent one-*ruleitems* discovered so far, we can easily obtain other frequent *ruleitems* that involve more than one attribute.

Broadley speaking, if itemsets a and b have passed the *minsupp* threshold, the intersection of the rowIds sets of a and b, results in a set of tuples where a and b happen to be together in the training data. Thus, class labels associated with the candidate itemset $a \wedge b$ can be easily located, from which the support and confidence can be found. Consider for instance itemsets $\langle(A_1, x_1)\rangle$ and $\langle(A_2, y_1)\rangle$ in Table 1, the following two sets represent the rowIds in which they occur, $\{1, 2, 3, 4, 8\}$ and $\{1, 3, 5, 6, 10\}$. We can determine the support of a new itemset, such as $\langle$ (A_1, x_1), $(A_2, y_1)\rangle$ by intersecting the rowId sets for itemsets $\langle(A_1, x_1)\rangle$ and $\langle(A_2, y_1)\rangle$. The cardinality of the resulting set $\{1, 3\}$ represents the support for itemset $\langle(A_1, x_1), (A_2, y_1)\rangle$, i.e. 2/10. If it survives the *minsupp* threshold, then we proceed by checking if it has sufficient number of occurrences when associated with its largest frequency class (that's if it is frequent) or else prune it. The support and confidence for frequent *ruleitems* are calculated by indexing their classes as we will discuss that in depth in Section 4.1.3.

4.1.3. Support and confidence computation for a ruleitem

In this section, we briefly explain how support and confidence for *ruleitems* are calculated using an example. We assume that all frequent one-*ruleitems* are determined and stored with their classes and locations. To find the support for a *ruleitem*, we locate the set of classes associated with it using its rowIds in the training data, and select the class with the largest frequency. Then by taking the cardinality of the set of the rowIds where the condition and the class of that *ruleitem* occur and divides that by the size of the training data set, we can obtain the *ruleitem* support.

The calculation of the confidence is done similarly except that the denominator of the fraction is the size of the set of the rowIds of the *ruleitem* condition instead of the size of the whole training data set. Frequent *ruleitems* are generated recursively from *ruleitems* having a smaller number of attributes, starting from frequent one-*ruleitems* derived in a single pass through the training data set. It should be noted that every time a frequent *ruleitem* is found, only the rule with the largest confidence is considered. In the case that a *ruleitem* is associated with two classes with identical confidence, the choice of the rule is random.

Attribute A_1

x_1	x_2	x_3
1	5	10
2	6	
3	7	
4	9	
8		

Attribute A_2

y_1	y_2	y_3	y_4
1	2	7	9
3	4	8	
5			
6			
10			

Fig. 2. Vertical data representation for training data in Table 4.

Consider the vertical representation shown in Fig. 2 for the training data set in Table 1. Assume that the *minsupp* and *minconf* have been set to 2 and 50%, respectively. During the scan, the survived frequent two-itemsets are shown in bold in Fig. 3 and all other itemsets and their rowIds are discarded. We only need to keep itemsets that are statistically significant at each level. Once these itemsets are identified at the first level, we check their supports and confidences simultaneously by locating classes that occur with their rowIds.

For example, for frequent itemset $\langle a_1, x_1\rangle$, we locate its classes in rowIds $\{1, 2, 3, 4, 8\}$, i.e. c_1 $\{1, 4, 8\}$ and c_2 $\{2, 3\}$. We choose the class with the largest frequency, i.e. $c_{1,}$ and divide the cardinality of the set $\{1, 4, 8\}$ by the size of the training data set, to obtain the support for *ruleitem* $\langle$ (a_1, x_1), $c_1\rangle$. If it holds enough support, we immediately calculate its confidence by dividing the size of rowIds of the *ruleitem's* largest class (the nominator of the *ruleitem* support), i.e. 3, with the size of the *ruleitem's* condition set, i.e. 5. For *ruleitem* $\langle$ (a_1, x_1), $c_1\rangle$ the support is 3/10 and the confidence is 3/5. In the case that the *ruleitem* passes the *minconf* threshold, we immediately generate it as a candidate rule in the classifier. There is no separate phase to calculate the confidences for all frequent *ruleitems* in MCAR, whereas the majority of current AC techniques produce frequent *ruleitems* in one step and find their confidences in a separate step.

4.2. Main differences between MCAR intersection method and other tid-list methods

There are some significant differences between our intersection method and other methods used in association rule mining algorithms (Savasere et al., 1995; Zaki & Gouda, 2003; Zaki et al., 1997) such as:

- The tid-lists method proposed in the association rule partitioning algorithm of (Savasere et al., 1995) has been used locally for each partition and not for the whole database. Thus, in order to come up with global frequent itemsets, the algorithm requires an additional database scan to evaluate whether local frequent itemsets in each partition are also globally frequents. Furthermore, the partition algorithm uses Apriori horizontal layout data representation, which suffers from many costs as mentioned earlier, whereas, our intersection method finds all frequent *ruleitems* from the whole data set once, relaxing the partition step and

x_1y_1	x_1y_2	x_1y_3
1	**2**	8
3	**4**	

x_2y_1	x_2y_2	x_2y_3
5		7
6		

Fig. 3. Possible frequent two-itemsets derived from Table 1.

takes advantage of the vertical layout representation, minimising the time needed for finding frequent *ruleitems*.

- Improvements upon the partition algorithm have been reported in an association rule mining approach, called Eclat (Zaki et al., 1997), where a vertical layout representation of the transactional database and a fast intersection of tid-lists are presented. However, in the original version of Eclat, only frequent itemsets of size ≥ 3 are generated. On the other hand, the intersection method presented in MCAR generates frequent *ruleitems* of size >1.
- A variation of Eclat algorithm that uses the so called diffsets has been presented in an algorithm called dEclat (Zaki & Gouda, 2003). Diffsets stores only the differences in the transactions identifiers (tids) of a candidate itemset from its generating frequent itemsets; rather than storing the complete tids of each itemset. dEclat uses diffsets rather than tid-lists to maintain less storage at each level. Our proposed intersection method uses rowIds similar to tids in Eclat, and not diffsets, therefore, the methodology of producing frequent *ruleitems* in MCAR is different than that of dEclat even though both algorithms use vertical data format.
- The tid-lists and diffsets intersection methods have been only used in association rule mining problems where the cardinality of an itemset tid-list divided by the total transactions in the database gives the support of that itemset. Whereas, the calculation of *ruleitems* support in classification problems is more complex due to not only the frequencies of the itemsets are considered but also the highest-class frequency associated with them. Consequently, additional calculations are required in the process of discovering frequent *ruleitems* in AC. We have extended the tid-lists intersection method proposed in vertical association rule mining algorithms to deal with classification benchmark problems.

4.3. Rule selection in the sorting process

Rules preference often determined based on some parameters, including support, confidence and cardinality. This section presents the additional parameters taken by the MCAR rule ranking method to discriminate between rules. AC techniques like CBA and CMAR rank rules mainly in terms of the confidence and support level. When several rules have identical confidences or supports, these techniques randomly choose one of the rules, which in some cases may degrade accuracy.

On the other hand, the MCAR tries to minimise random selection whenever there is a choice between tie rules by imposing additional constraints. The rule ranking method introduced in this section and shown in Fig. 4 breaks ties between rules having high strength and statistically representative by looking into classes associated with them. Moreover, it uses other tie breaking conditions such as rules cardinality and items ordering in the training data set (Li et al., 2001).

To show the need for extra parameters to discriminate between tie rules, consider the following example: If someone mines the 'balance-scale' data set, which has been downloaded from (WEKA, 2000) with a *minsupp* of 5% and *minconf* of 40% using the CBA algorithm (Liu et al., 1998), the resulting classifier consists of 15 rules, all of which have identical supports and the top 4 also have identical confidences, leaving no way for CBA to discriminate between them. Moreover, the 15 rules extracted by CBA have the same length, and therefore other rule ranking methods that employ the cardinality as a tie breaking such as CMAR (Li et al., 2001) can not favour one rule over another in this case.

A more serious case is the 'tic-tac' data set, if we mine it using the same support and confidence, there will be 340 potential rules with the same confidence and 270 of them have identical support too. When the support is lowered further to 2 or 1%, there could be a huge number of rules with identical support and confidence, therefore, additional tie breaking conditions are essential to minimise random selection. Also since classification data sets are often highly dense, then there can be many rules with similar support, confidence and cardinality, which make it hard for most current AC algorithms to distinguish between them in the rule sorting process.

Though, MCAR adds upon the previous approaches by looking at class labels frequencies associated with the tied

Given two rules, r_a and r_b,r_a precedes $r_b(r_a\rangle r_b)$ if:

1. The confidence of r_a is greater than that of r_b.
2. The confidence values of r_a and r_b are the same, but the support of r_a is greater thanthat of r_b.
3. Confidence and support values of r_a and r_b are the same, but r_a has fewer conditions in its left hand side thanof r_b.
4. Confidence, supportand cardinality of r_a and r_b are the same, but r_a is associated with a more representative class than that of r_b.
5. All above criteria are identical for r_a and r_b, but r_a generated from an items and columns that have higher order than that of r_b.

Fig. 4. Rule selection in MCAR.

rules in the training data set, and selects the rule associated with the largest frequency class. In addition, it looks at the position of the items and columns used to produce each rule in the database and favours rules that have a higher order. We will show in Section 5 the effectiveness of the proposed rule sorting method on the quality of the produced classifiers.

4.4. Classifier builder and pruning

A rule is significant if and only if it covers at least one training instance. After the rules have been generated, an evaluation step tests each rule in order to remove rules which fail to classify at least one training instance and place high confidence rules at the top of the classifier. At each step in the rule evaluation phase, all rows correctly classified by the selected rule will be deleted from the training data set. Whenever a rule does not classify any rows of the data, it will be removed from the rules set because a higher ranked rule has correctly classified its instances. This pruning should remove many redundant rules. Fig. 5 represents the classifier builder algorithm used by MCAR, which has three main steps:

1. Rank rules produced in R based on the sorting procedure shown in Fig. 4 (line 1). This step is important because we only look for large confidence and support rules to keep in the classifier.
2. For each rule r, find all instances that match r body by indexing. Since, we hold the rowIds for each rule in arrays during the learning phase, locating rule items in the training data by using their rowIds, is a straightforward task. If r correctly covers at least one training instance (line 4), then it is marked as a candidate rule (line 6), all instances associated with r are removed (line 7) from the training data and r is inserted in the classifier (line 10). We continue until we finish all rules, at that point, we choose a default class (line 14).
3. Prune redundant rules to improve the quality of the classifier as follows: A rule R_1:I$\rightarrow$c is said to be more general than R_2:I$'\rightarrow$c, if and only if I is subset of I$'$ and $R_1 > R_2$. This pruning removes specific rules having less confidence values than their subsets of general rules, which significantly reduces the number of rules in the resulted classifier and minimises redundancy.

```
Input: set of created rules (R) and training data set (T)
Output: classifier (Cl)

1 R' = sort(R);
2 for each rule r ∈ R' in sequence do
3 find all instances that match r body.
4    if r classifies at least a single instance
5  begin
6        mark r as a candidate rule;
7        delete all instance in T contain r body
8    end if
9               if r is marked then
10              insert r at the end of Cl;
11  end
12   If T.size > 0 then
14    select the majority class as a default class form
       the remaining unclassified instance in T
15   else
16      select the majority class as a default class from
        the current Cl and add it to Cl
```

Fig. 5. MCAR classifier builder algorithm.

The classifier builder ensures that each training instance is covered by at most one rule, which has the highest precedence among all rules applicable to it. Furthermore, there are no useless rules in the MCAR classifier since every rule correctly covers at least one training instance. This approach is similar to the CBA classifier builder as each rule in CBA also covers at least one training instance. However, the way MCAR builds the classifier by locating training instances is more efficient than that of CBA due to abounding going through the training data set multiple times.

To clarify, one needs to go though the training data more than once while building the classifier in CBA to determine the subset of rules that form the classifier. For each training instance *t*, CBA finds the highest precedence rule that correctly classifies *t*, (r_c) and the highest precedence rule that incorrectly classifies *t*, (r_i). If $r_c > r_i$, CBA assigns r_c to *t*, but, if $r_i < r_c$, CBA stores *t* along with its r_c and r_i in a data structure object and goes through the training data again to determine which rule to assign to *t* by looking at those r_i which have been used by other training instances as r_c. This is the reason why CBA classifier builder requires more than on scan, whereas, the classifier procedure of MCAR locates training instances for each rule by using their items rowIds stored in the arrays. Consequently, there is no need to scan the training data in order to find applicable training instances for each rule, which substantially reduces the time needed to construct the classifier.

4.5. Prediction of test instances

In prediction the classes of a test data set, let *R* be the set of generated rules and *Ts* be the set of test data objects. The basic idea of the proposed method as shown in Fig. 6 is to choose the best rule among a set of high confidence, representative and general rules in *R* to cover *Ts*. In classifying a test object (line 1), the proposed algorithm uses a simple method, which states that the first rule in the set of ranked rules that matches the test object condition classifies it (line 5). In a case where no rule fully matching the test object condition, MCAR prediction procedure seeks for the highest precedence rule that matches part of the test object body and applies it (line 7). In cases where no rule matches the test object condition, the default class will be assigned to the test object (line 8).

This classification process ensures that only the highest ranked rules classify test objects. The prediction method presented in MCAR is similar to CBA prediction method where an instance is fired by only one rule to overcome conflict decisions. Except that, MCAR tries to find an exact match to the test object in the classifier instead on taking the highest precedence rule that partially matches the test instance. When no rule fully matching the test instance is found, then MCAR takes on the highest precedence rule as CBA does.

4.6. MCAR features

The proposed algorithm has the following distinguishing features over other AC algorithms:

- Other AC techniques often use horizontal layout representation where multiple passes are required to discover frequent *ruleitems*, Alternatively, and to the best of our knowledge, MCAR is the first AC method that uses vertical layout representation and a recursive technique based on intersection to discover rules, which requires only one scan. It should be noted that, a traditional decision tree algorithm called SPRINT (Shafer, Agrawal, & Mehta, 1996) uses a similar data format to vertical layout to store attribute values called attribute lists. However, SPRINT does not use fast intersections of rowIds to discover frequent *ruleitems*, instead it uses an information gain approach to build decision trees similar to (Frank & Witten, 1998; Quinlan, 1993)
- Most AC techniques, e.g. CBA, CMAR, locate training instances matching each potential rule by going through the training data more than one time during building the classifier. MCAR on the other hand, avoids going through the training data and locates training instances for each rule using their rowIds.
- MCAR employs a rule ranking technique, which uses two new conditions to break ties between rules. Our rule selection method looks at the class distribution frequencies, items and columns ordering for each rule in the training data and favours rules that are associated with the dominant class and have higher order.
- There are several association rule mining algorithms that discover frequent itemsets and rules simultaneously, i.e. (Zaki, 1999). MCAR takes the principle of such approaches and utilises it in AC to find frequent *ruleitems* and produce rules in one main step. Other AC algorithms, including (Antonie & Zaïane, 2004; Baralis & Torino, 2000; Liu et al., 1998; Liu et al., 2003) find frequent *ruleitems* in one step and generate rules in a separate step, requiring more computational time.

```
Input: Classifier (R) and test data set (Ts)
Output: Prediction error rate Pe

Given a test data (Ts), the classification process works as
follow:

1 For each test instance ts Do
2 For each rule r in set of ranked rules R Do
3   Find all applicable rules that match ts body
     and store them in Tr
4   If Tr is not empty
      begin
5       If there exists a rule r that fully matches ts
        condition
6          assign it to the ts
7       else assign the highest precedence rule class to ts
8     end if
9   else assign the default class to ts
10 empty Tr
11 end
12 end
13 compute the total number of errors of Ts;
```

Fig. 6. MCAR prediction algorithm.

5. Experimental results

Experiments on 20 different data sets from the UCI data collection (Merz & Murphy, 1996) were conducted using stratified 10-fold cross validation (Witten & Frank, 2000). Three popular classification techniques: decision trees (C4.5) (Quinlan, 1993), RIPPER (Cohen, 1995) and CBA (Liu et al., 1998) have been compared to our proposed algorithm in terms of error rate, efficiency, rule features and number of rules. The choice of such learning methods is based on the different strategies they use to generate the rules.

We have observed in the experiments of CBA that its authors have used the same random seed when partitioning the training data in cross validation, which results in an optimistic measurement of accuracy. Thus, if a user mines a training data set *A* 10 different times, the resulting error rate on *A* for the 10 times will be identical. Though in cross validation, the training data should be randomly split, and therefore the resulting error rate should be slightly different every time.

We ran the CBA against ‘contact-lenses’ data set twice, once with the original version that been downloaded from (Merz & Murphy, 1996), and once with a shuffled version. The resulting error rate on the original data set was 15.00%, but, when we shuffled the data before mining, we observed that the error rate has increased to 20.00%. In this case the CBA system used a random seed that produced somewhat optimistic results. Using a truly (pseudo) random row ordering gives a more accurate evaluation of the classification accuracy, for this purpose, we have implemented a simple shuffling method that uses a different random seed for each run. We simply change the location of the training objects randomly according to the size of the training data set by a factor specified by the end-user. This ensures that a particular ordering of the rows in the training data will not impact the error rate. The MCAR error rate shown in Table 2 is the average error rate obtained in 10 different runs for each data set. In all experiments, data sets were shuffled before they have been mined.

Table 2 gives the error rates of CBA, C4.5, RIPPER and our proposed algorithms obtained on the 20 data sets. The experiments of C4.5 and RIPPER algorithms were conducted using *Weka* software system (WEKA, 2000). *Weka* stands for Waikato Environment for Knowledge Analysis, which is an open java source code for the machine learning community that includes implementations of several data mining tasks such as classification, clustering, association rule mining and regression. CBA experiments were conducted using an implementation version provided by the authors of (CBA, 1998) and MCAR was implemented in Java.

From our experiments, we observed that the classifiers derived when the support threshold was set between 2 and 5% achieved good accuracy, and most often better than that of C4.5, RIPPER and CBA, and following the experiments, the *minsupp* was set to 5%. The confidence threshold, on the other hand, has a smaller impact on the behaviour of any AC method, and it has been set in our experiments to 35%. The results shown in Table 2, where the least error rate for each data set is in bold, indicate that our proposed algorithm outperforms the other rule learning techniques on the majority of the data sets.

One of the main reasons for this appears to be that MCAR rule ranking method reduces the chance of random selection between tie rules, which in most cases lead to a better decision. Shortly, we will show the impact of the additional tie breaking parameters used by MCAR rule selection method on the quality on the resulting classifiers. Another reason for the slight decrease of error rate appears to be the less pruning utilised by MCAR than CBA, RIPPER and C4.5, which often leads to the production of larger classifiers than the other learning algorithms. The won-loss-tied record of the proposed algorithm against CBA in term of error rate is 11-8-1. The won-loss-tied record of the proposed method against C4.5 and RIPPER algorithms in term of error rate are 13-7-0 and 16-3-1, respectively.

The error rate figures revealed also that CBA outperforms RIPPER and C4.5 algorithms, which support results reported in other research works, e.g. (Li et al., 2001; Liu et al., 1998; Yin and Han, 2003). These results provide facts that AC algorithms generate more accurate classifiers than decision trees and rule induction approaches. However, they have their own drawbacks such as the exponential growth of rules, which is not addressed in this paper.

To validate the significance of the proposed rule sorting method, Table 3 shows the number of times each condition of the MCAR rule selection method has been used to break between tie rules for 12 highly dense classification problems. Columns 'No. of rules with the same Conf.' and 'No. of rules with the same Conf. & Supp.' stand for the confidence and support conditions, respectively. Column 'No. of rules with the same Conf., Supp. & Cardinality' corresponds to the cardinality of the rule condition and column 'No. of rules with the same Conf., Supp., Cardinality & Class Freq.' represents the largest frequency class associated with each rule. The last column 'RowOrd & ColOrd' indicates the row and column ordering for items contained in the rules. The "RowOrd & ColOrd" combined gives a better measure for favouring rules than random selection. We have used a *minsupp* of 2% and a *minconf* of 40% to produce the numbers shown in Table 3. These numbers represent the potential rules tested by the MCAR algorithm during the ranking process and before building the classifier or performing any pruning.

Table 2
Error rate of MCAR, CBA, C4.5 and RIPPER algorithms using 10-fold cross validation

Data	Size	Classes	C4.5	RIPPER	CBA	MCAR
Cleve	303	2	23.77	22.45	16.87	17.13
Breast-w	699	2	5.44	4.58	4.16	3.52
Diabetes	768	2	26.18	23.96	24.66	22.18
Glass	214	7	33.18	31.31	30.11	30.33
Iris	150	3	4.00	5.34	6.75	4.68
Pima	768	2	27.22	26.70	24.51	21.46
Wine	178	3	5.62	7.31	1.67	2.89
Austral	690	2	14.79	14.79	14.64	12.57
German	1000	2	29.10	27.80	27.43	27.90
Labor	57	2	26.32	22.81	5.01	12.81
Tic-tac	958	2	16.29	3.03	0.00	0.00
Led7	3200	10	26.44	30.47	28.26	28.76
Heart-s	294	2	18.71	21.77	20.80	18.86
Lymph	148	4	18.92	22.98	23.62	23.98
Vote	435	2	11.73	12.65	13.09	11.30
Zoo	101	7	6.94	14.86	4.04	2.22
Balance-scale	625	3	35.68	25.44	34.34	22.46
Primary-tumor	339	23	58.41	65.20	74.89	58.90
Mushroom	8124	2	0.23	0.10	8.71	2.44
Contact-lenses	24	3	16.67	25.00	20.00	25.00

Table 3
Number of times each condition in the rule ranking method does not break tie between potential rules

Data	No. of rules with the same Conf.	No. of rules with the same Conf. & Supp.	No. of rules with the same Conf., Supp., & Cardinality	No. of rules with the same Conf., Supp., Cardinality & Class Freq.	RowOrd and ColOrd
Autos	2660	2492	2117	1683	181
Glass	759	624	409	245	7
Lymph	11019	10775	10217	9595	2381
Cleve	17092	16289	14647	12942	469
Tic-tac	2297	2047	1796	1541	278
Diabetes	252	91	15	3	0
Breast	3471	2980	2217	1643	75
Vote	7755	6802	5126	4013	207
Heart	4791	4267	3383	2562	145
Pima	252	91	15	3	0

The figures obviously provide a clear picture that additional tie breaking constraints beside support and confidence are vital as they have been used heavily to distinguish between rules in the rule sorting process for the 12 benchmark problems. For the ‘Cleve’ data set for instance, there are 17092 potential rules with similar confidence, in which 16289 of them have identical support. From these, there are 14647 with the same cardinality and 12942 from the 14647 are associated with classes that have the same frequency in the training data set.

The frequent use of the two conditions proposed in the MCAR sorting method suggests the validity of the hypothesis, which states the more constraints imposed to discriminate between tie potential rules, the more random selection is minimised, which certainly should positively increase the accuracy of the classifiers.

To show the effectiveness of the rule sorting method on the quality of the classifiers produced, we conduct a large number of experiments with reference to accuracy in order to compare between the proposed rule sorting method and two other popular existing rule sorting methods in AC, which are CBA and CMAR ones. We have implemented the three rule sorting methods in Java within the MCAR algorithm. We have used 10 fold cross validation in the experiments and each value in Table 4 represents an overage of ten cross validation runs since, we utilise different random seed when partitioning the training data set in cross validation.

Table 4
Impact of the three rule sorting method on the accuracy of MCAR algorithm

Data	CBA	CMAR	MCAR
Cleve	82.44	82.16	81.35
Breast-w	94.61	95.11	96.32
Diabetes	76.90	77.05	77.18
Glass	67.76	68.79	69.97
Pima	77.16	77.34	77.11
Tic-tac	99.76	99.77	100.00
Led7	70.95	71.07	71.00
Heart-s	81.30	81.87	82.14
Lymph	79.10	77.13	78.57
Vote	88.86	88.17	87.70
Zoo	95.38	97.73	97.78
Contact-lenses	72.93	73.54	75.54
Average	82.26	82.48	82.88

Table 4 indicates the accuracy of the classifiers generated by MCAR algorithm when each rule sorting method is used on the 12 benchmark problems. The figures show a slight improvement of the proposed rule sorting method over existing AC rule sorting methods. In particular, our hybrid rule selection method achieved on average +0.63 and +0.41% improvements with reference to accuracy on the 12 benchmark problems over that of CBA and CMAR rule sorting methods, respectively. This gives evidence that the additional constraints imposed to break between tie rules by MCAR slightly improve the predictive power of the resulting classifiers.

We compared the processing time taken C4.5, RIPPER, CBA and MCAR to build the classifier on the 18 data sets in order to compare scalability and efficiency. We would like to validate whether the classifier builder presented in MCAR algorithm reduces the training time if compared with that of CBA. All the runtime experiments were conducted on Pentium IV 1.7 Ghz, 256 RAM machine. Table 5 shows the runtime in seconds obtained in the experiments. The runtime numbers revealed that MCAR is faster than CBA in most cases.

Table 5
Runtime for building the classifiers in seconds

Data	C4.5	RIPPER	CBA	MCAR
Cleve	0.020	0.050	0.890	0.300
Breast-w	0.030	0.200	1.430	0.670
Diabetes	0.160	0.170	1.390	0.770
Glass	0.030	0.130	0.650	0.210
Iris	0.020	0.020	0.480	0.150
Pima	0.060	0.220	1.180	0.770
Wine	0.020	0.030	0.760	0.180
Austral	0.020	0.020	1.800	0.690
German	0.130	0.340	2.710	1.000
Labor	0.020	0.020	0.050	0.050
Tic-tac	0.020	0.420	1.360	0.515
Led7	0.130	1.580	5.670	0.689
Heart-s	0.005	0.005	0.500	0.031
Lymph	0.005	0.020	0.410	0.281
Vote	0.005	0.030	0.970	0.960
Zoo	0.188	0.005	0.020	0.430
Balance-scale	0.013	0.005	0.440	1.110
Contact-lenses	0.005	0.005	0.005	0.005

The intersection method that MCAR employed to find frequent *ruleitems* is significantly an advantage. Moreover, abounding going through the training data set to locate instances applicable to each rule in the training phase reduces the I/O time. The training time results indicated that traditional classification algorithms like decision trees and RIPPER are faster than AC methods. This is due the simplicity in the way they construct the classifier, based on heuristic search, whereas, AC algorithms use exhaustive search methods adapted from association rules, which necessitates higher computational complexity. It should be noted that the classification data sets used in the experiments are smaller in size than market basket databases used in association rule mining, therefore a slight improvement in runtime for a small data set in classification can be considered a notable improvement.

A deeper analysis of the rules produced by MCAR and CBA has been conducted to compare their effects on the accuracy. Let us consider the classifiers derived by MCAR, and CBA from three benchmark problems, i.e. 'contact-lenses', 'zoo' and 'balance-scale'. Figs. 7(a,b), 8(a,b) and 9(a, b) represent the support and confidence values derived by CBA and MCAR on the above mentioned data sets using *a minsupp* of 5% and a *minconf* of 35%. After analysing the rules generated, it was found that there is consistency in the rule features between CBA and MCAR. For example, 15 of the generated rules from 'balance-scale' and four from 'contact-lenses' are identical in CBA and MCAR classifiers. However, the MCAR method derived more rules than CBA on all data sets, with all rules having confidence as large or greater for MCAR than CBA. For instance, the classifier derived by CBA algorithm on the 'balance-scale' has an error rate of 34.34% data set. By comparison, the MCAR classifier has one more rule, giving a much reduced 11.88% error rate.

Table 6 indicates the number of rules extracted from the rule learning algorithms where association rules based techniques extract more rules than heuristic classification algorithms. In fact, Table 6 shows that MCAR often derives more number of rules than CBA due to the less pruning employed, whereas CBA utilises the decision tree pessimistic error pruning. This results in learning quite more rules than CBA, and thus explains the reason behind extracting more rules.

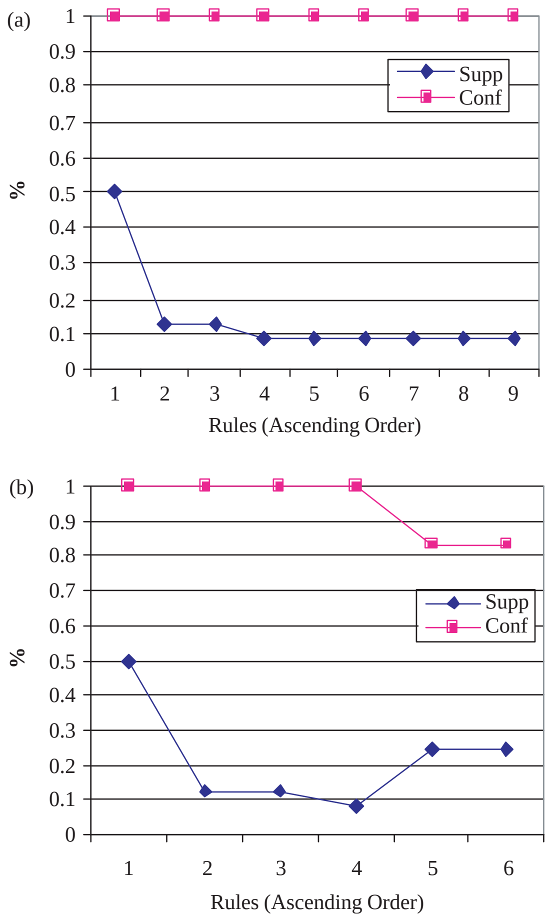


Fig. 7. (a) Support and confidence values of MCAR classifier for contact-lenses data. (b) Support and confidence values of CBA classifier for contact-lenses data.

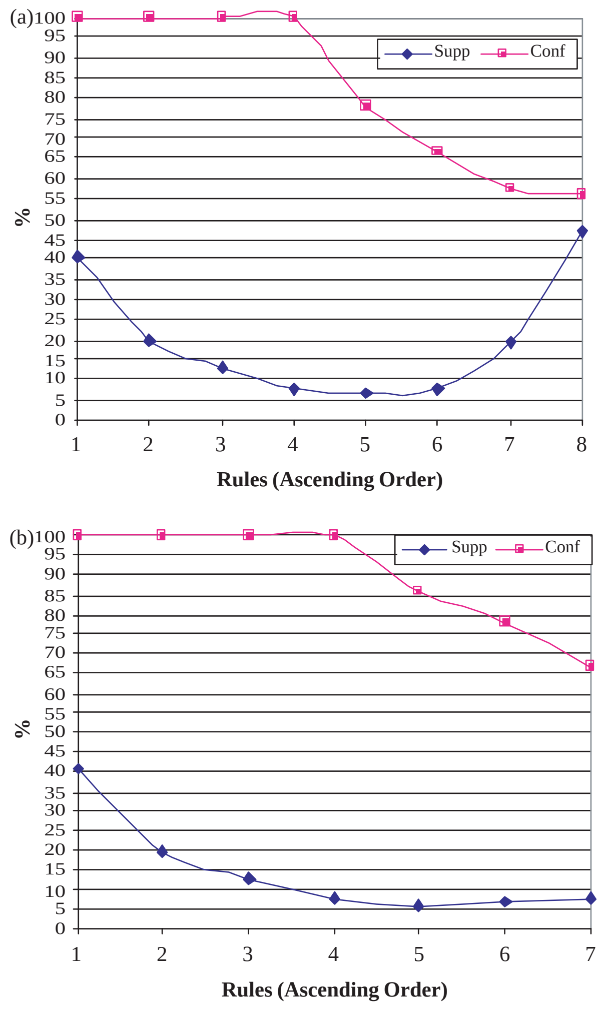
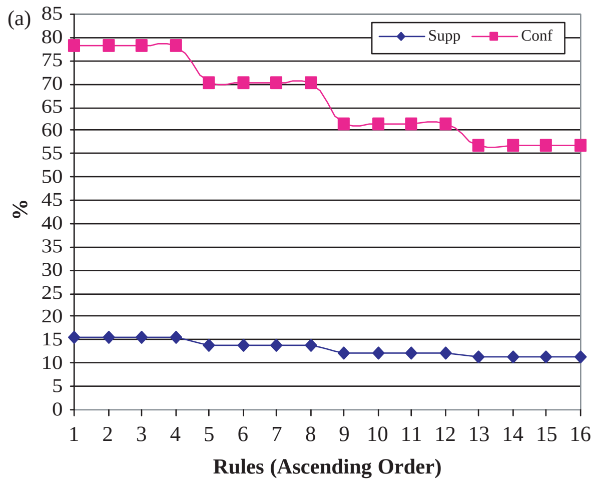


Fig. 8. (a) Support and confidence values of MCAR classifier for zoo data. (b) Support and confidence values of CBA classifier for zoo data.



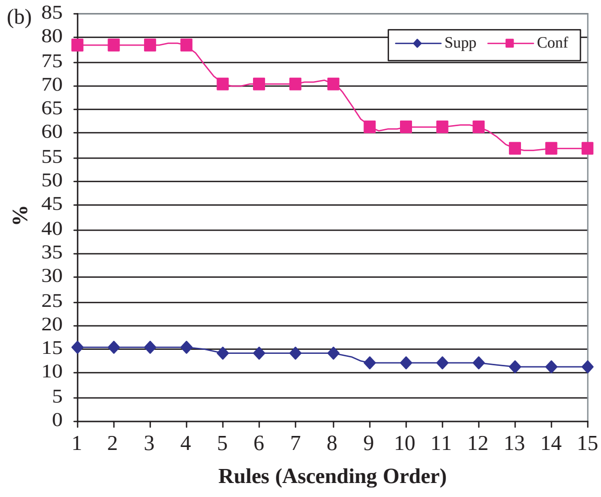


Fig. 9. (a) Support and confidence values of MCAR classifier for balance-scale data. (b) Support and confidence values of CBA classifier for balance-scale data.

6. Conclusions

In this paper, the problem of using association rule approaches in classification has been investigated. The outcome is a new effective AC algorithm that has the following features over other existing techniques:

- MCAR extends the fast intersection method used in vertical association rule mining approaches to handle classification benchmark problems. The intersection method used by the MCAR algorithm requires only one pass over the training data, consuming significantly less storage and I/O time than other multi-pass approaches.
- MCAR presents a rule ranking technique, which uses two new conditions, in which one of them considers the class distribution frequencies to discriminate between tied rules. The other condition looks at the items ordering and column ordering in the training data. All of these conditions have been used to minimise random selection when a choice between two or more rules occurs in the rule ranking process.

Table 6
Number of rules of MCAR, CBA, C4.5 and RIPPER algorithms using 10-fold cross validation

Data	C4.5	RIPPER	CBA	MCAR
Cleve	30	5	72	100
Breast-w	14	6	45	61
Diabetes	20	4	36	66
Glass	30	8	36	66
Iris	5	4	18	31
Pima	26	3	36	66
Wine	5	4	11	12
Austral	9	4	121	179
German	103	3	157	225
Labor	5	4	17	16
Tic-tac	95	9	25	27
Led7	37	19	53	193
Heart-s	2	2	22	31
Lymph	12	6	38	49
Vote	4	4	40	85
Zoo	10	6	7	8
Balance-scale	33	17	15	16
Primary-tumor	23	7	1	31
Mushroom	44	14	38	53
Contact-lenses	4	3	6	9

- In building a classifier, most AC techniques locate training instances matching each potential rule by going through the training data more than one time. Alternatively, and to reduce CPU time, locating training instances applicable to a selected rule in MCAR is carried out by using the rowIds of the items in the selected rule.
- MCAR finds frequent *ruleitems* and produce rules in the same step, whereas most other AC methods like CBA, L^3 and CMAR find frequent *ruleitems* in one step and generate rules in a separate step, requiring more computational expenses.

Performance studies on 20 data sets from UCI data collection indicated that our proposed algorithm is highly competitive when compared with traditional classification algorithms such as RIPPER and C4.5 in term of prediction accuracy. Furthermore, MCAR scales well if compared with popular AC like CBA with regards to prediction power, rules features and efficiency. Experiments using 12 highly correlated classification problems revealed that adding more constraints to distinguish between ties rules improve the quality of the resulting classifiers. Particularly, the rule sorting method of MCAR improves the accuracy on average +0.61 and +0.41% more than of that CBA and CMAR sorting procedures.

MCAR produced classifiers with slightly more rules than current AC techniques, resulting in reduced error rate. However, for some benchmark problems a post pruning method may be beneficial to reduce the number of rules derived. Finally, the experiments revealed that changing locations for some training instances before mining may greatly deteriorates the quality of the resulting classifier. In the near future, we will investigate the extraction of multiple class labels using association rule discovery for a wide range of application problems.

References

Agrawal, R., & Srikant, R. (1994). Fast algorithms for mining association rule. *Proceedings of the 20th international conference on very large data bases* pp. 487–499.

Ali, K., Manganaris, S., & Srikant, R. (1997). Partial classification using association rules. In D. Heckerman, H. Mannila, D. Pregibon, & R. Uthurusamy (Eds.), *Proceedings of the third international conference on knowledge discovery and data mining*, 115–118.

Antonie, M., & Zaïane, O. (2004). An associative classifier based on positive and negative rules. *Proceedings of the 9th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery* (pp. 64–69). Paris, France.

Antonie, M., Zaïane, O., & Coman, A. (2003). Associative classifiers for medical images. *Lecture notes in artificial intelligence 2797, mining multimedia and complex data*. New York: Springer pp. 68–83.

Baralis, E., & Torino, P. (2002). A lazy approach to pruning classification rules. *Proceedings of the 2002 IEEE ICDM'02* pp. 35.

Brin, S., Motwani, R., Ullman, J., & Tsur, S. (1997). Dynamic itemset counting and implication rules for market basket data. *Proceedings of the 1997 ACM SIGMOD international conference on management of data* pp. 265–276.

CBA (1998). http://www.comp.nus.edu.sg/~dm2/p_download.html.

Cendrowska, J. (1987). PRISM: An algorithm for inducing modular rules. *International Journal of Man–Machine Studies*, *27*(4), 349–370.

Cohen, W. (1995). Fast effective rule induction. *Proceedings of the 12th international conference on machine learning*. Los Altos, CA: Morgan Kaufmann pp. 115–123.

Fayyad, U., & Irani, K. (1993). Multi-interval discretisation of continues-valued attributes for classification learning. *Proceedings of IJCAI* pp. 1022–1027.

Frank, E., & Witten, I. (1998). Generating accurate rule sets without global optimisation. *Proceedings of the fifteenth international conference on machine learning*. Madison, WI: Morgan Kaufmann pp. 144–151.

Furnkranz, J. (1999). Separate-and-conquer rule learning. *Artificial Intelligence Review*, *13*(1), 3–54.

Han, J., Pei, J., & Yin, Y. (2000). Mining frequent patterns without candidate generation. In *Proceeding of the 2000 ACM SIGMOD international conference on management of data* (pp. 1–12). Dallas, TX, May.

Huange, J., Zhao, J., & Xie, Y. (1997). Source classification using pole method of AR model. *IEEE ICASSP*, *1*, 567–570.

Li, W. (2001). Classification based on multiple association rules, MSc thesis. Simon Fraser University, April.

Li, W., Han, J., & Pei, J. (2001). CMAR: Accurate and efficient classification based on multiple-class association rule. *Proceedings of the ICDM'01* (pp. 369–376). San Jose, CA.

Liu, B., Hsu, W., & Ma, Y. (1998). Integrating classification and association rule mining. *Proceedings of the KDD* (pp. 80–86). New York, NY.

Liu, B., Ma, Y., & Wong, C.-K. (2001). Classification using association rules: weakness and enhancements. In V. Kumar (Ed.), *Data mining for scientific applications*.

Merz, C., & Murphy, P. (1996). *UCI repository machine learning databases*. Irvine, CA: University of California.

Park, J., Chen, M., & Yu, P. (1995). An effective hash-based algorithm for mining association rules. *Proceedings of the ACM SIGMOD* (pp. 175–186). San Jose, CA.

Pazzani, M., Mani, S., & Shankle, W. (1993). Beyond Concise and colourful: Learning intelligible rules. *Proceeding of the KDD*. Menlo Park, CA: AAAI Press pp. 235–238.

Quinlan, J. (1987). Generating production rules from decision trees. *Proceedings of the 10th international joint conferences on artificial intelligence*. Los Altos, CA: Morgan Kaufmann pp. 304–307.

Quinlan, J. (1993). *C4.5: Programs for machine learning*. San Mateo, CA: Morgan Kaufmann.

Savasere, A., Omiecinski, E., & Navathe, S. (1995). An efficient algorithm for mining association rules in large databases. *Proceedings of the 21st conference on very large databases VLDB '95, Zurich, Switzerland, Septemper 1995* pp. 432–444.

Shafer, J., Agrawal, R., & Mehta, M. (1996). SPRINT: A scalable parallel classifier for data mining. *Proceedings of the 22nd international conference on very large data bases* (pp. 544–555). Bombay, India, September 1996.

Wang, K., He, Y., & Cheung, D. (2001). Mining confidence rules without support requirements. *Proceedings of the tenth international conference on Information and knowledge management* (pp. 89–96). Atlanta, Georgia.

WEKA (2000). Data mining software in Java: http://www.cs.waikato.ac.nz/ml/weka.

Witten, I., & Frank, E. (2000). *Data mining: Practical machine learning tools and techniques with java implementations*. San Francisco, CA: Morgan Kaufmann.

Yin, X., & Han, J. (20030. CPAR: Classification based on predictive association rule. *Proceedings of the SDM* (pp. 369–376). San Francisco, CA.

Zaki, M. (1999). Parallel and distributed association mining: A survey. In IEEE concurrency, special issue on parallel mechanisms for data mining. December, 1999, Vol. 7. no. 4, (pp14–25).

Zaki, M., & Gouda, K. (2003). Fast vertical mining using diffsets. *Proceedings of the ninth ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 326–335). Washington, DC.

Zaki, M., Parthasarathy, S., Ogihara, M., & Li, W. (1997). New algorithms for fast discovery of association rules. *Proceedings of the third KDD conference* pp. 283–286.